Interfaces et graphes

Pré-requis : notion de graphe (explications au tableau)

Certaines questions de ce TP sont plus difficiles et pourraient éventuellement être sautées. Elles sont indiquées par (*) et (**).

1 Observation de l'implémentation d'une interface graphique

Exécuter le script graphes.py
Deux fichiers au format .grf, propre à cette application sont fournis : test1.grf et test2.grf

Question 1 : À l'aide de ces fichiers test et des fonctionnalités du script, expliquer ce qu'est un graphe eulérien.

Question 2 : Créer le graphe suivant et l'enregistrer :
Question 3 : En ajoutant une arête transformer le graphe précédent en graphe eulérien.

Question 4 : Observer à l'aide d'un bloc-notes le fichier créé à la Question 2. Quelle est sa structure ?

Question 5 : Observer les lignes 490-541 : supprimer l'onglet "Aide" dans le menu "Aide"

Question 6 : Observer les lignes 103-106 : modifier le comportement du bouton "donothing" de sorte à ce qu'il affiche le texte "ce bouton ne fait rien"

Question 7 : (*) Observer les lignes 112-134 : modifier le comportement du bouton "donothing" de sorte à ce qu'il se ferme lorsqu'on appuie dessus.

Question 8 : (**) Lorsqu'on essaye de modifier une arête, si on oublie de sélectionner une arête, le script produit une erreur. En s'inspirant par exemple de la ligne 446, résoudre ce problème. Dans la mesure du possible, une interaction avec l'utilisateur serait appréciable.

2 Prise en main de la classe Graphes

On ne s'intéresse désormais qu'à l'implémentation de la classe Graphes. On pourra donc se contenter d'ouvrir le fichier graphessimple.py

2.1 Création d'un graphe

Exécuter dans la console le code suivant :

```
S1=Sommet("Sup")
S1=Sommet("Spé")
S3=Sommet("École")
monG=Graphe("",[S1,S2,S3])
monG.add_edge(S1,S2,1)
monG.add_edge(S2,S3,0.5)
print(S1)
print(monG)
```

2.2 Méthode __repr__

Question 9 : Modifier print(G) pour qu'il y ait double saut de lignes entre le nom du graphe et la liste de ses sommets

Question 10 : (*) Modifier la classe Sommet de sorte à ce que print(S1) renvoie le nom du Sommet S1.

2.3 Création de méthodes sur les graphes

Question 11 : Créer dans la classe Graphe une méthode degre(S) qui renvoie le degré d'un sommet S. Le degré d'un sommet dans un graphe est le nombre de sommets auquel il est relié.

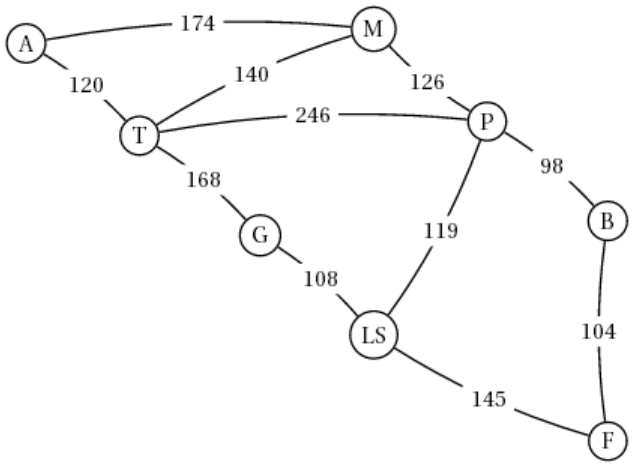
Question 12 : Créer une méthode connexe_est_eulerien() qui teste si un graphe connexe est eulérien. Un graphe est dit connexe si il existe un chemin qui relie n'importe quels deux sommets (il est en "un seul morceau") entre eux. Les graphes seront ici supposés connexes. Un graphe connexe est eulérien si tous ses sommets sont de degré pair.

3 Algorithme de Dijkstra

L'algorithme de Dijkstra décrit ci-dessous est un algorithme servant à trouver le plus court chemin entre deux sommets sur un graphe non orienté où les poids associés aux arêtes sont tous strictement positifs. Il est par exemple utilisé dans le calcul des itinéraires routiers, le poids des arêtes pouvant être la distance (pour le trajet le plus court), le temps estimé (pour le trajet le plus rapide) ou la consommation de carburant et le prix des péages (pour le trajet le plus économique).

```
Fonction Dijkstra (nœuds, fils, distance, début, fin)
    Pour n parcourant nœuds
        n.parcouru = -1
        n.précédent = 0
    début.parcouru = 0
    pasEncoreVu = nœuds
     Tant que pasEncoreVu!= liste vide
        n1 = minimumPositif(pasEncoreVu)
        pasEncoreVu.enlever(n1)
        Pour n2 parcourant fils(n1) // Les nœuds reliés à n1 par une arête
            Si n2.parcouru<0 ou n2.parcouru > n1.parcouru + distance(n1, n2)
                n2.parcouru = n1.parcouru + distance(n1, n2)
                n2.précédent = n1
    chemin = liste vide
    n = fin
    Tant que n!= début
        chemin.ajouterAvant(n)
        n = n.précédent
    chemin.ajouterAvant(début)
    Retourner chemin
```

Question 13 : Le graphe suivant représente les distances entre plusieurs villes d'Italie : Aoste, Milan, Parme, Turin, Gênes, La Spezia, Bologne et Florence. Chaque ville est désignée par son initiale. Appliquer à la main l'algorithme de Dijkstra pour trouver le plus court chemin entre Aoste et Florence.



Question 14 : Implémenter l'algorithme de Dijkstra.

Question 15 : Tester votre résultat sur le graphe de la Question 13.